P3XU-E501-02Z0

エンベデッドコンピュータAR2000シリーズ
フロントオペレーションモデル
RAS制御　API説明書（Windows編）

このページは空白です。

はじめに

このたびは、弊社のエンベデッドコンピュータ AR2000 シリーズをお買い求めいただき、誠にありがとうございます。
本製品は、省スペース設計の組み込み用コンピュータです。
本書は、本製品の RAS 制御 API 機能詳細（Windows 編）について説明しています。
本書をご覧いただき、本製品を正しくお使いくださいますよう、お願いいたします。

2007 年 5 月

Microsoft および Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。
Linux は、Linus Torvalds 氏の米国およびその他の国における登録商標または商標です。
Red Hat および Red Hat をベースとしたすべての商標とロゴは、米国およびその他の国における Red Hat, Inc. の商標または登録商標です。
Adobe、Reader は、Adobe Systems Incorporated（アドビシステムズ社）の登録商標です。
その他の社名および商品名は各社の登録商標または商標です。

All Rights Reserved, Copyright © PFU LIMITED 2006-2007

本書の読み方

本書の構成内容、本書での表記に関する注意事項などについて説明します。

●マニュアル体系

本製品には以下のマニュアルが用意されています。必要に応じてお読みください。

『取扱説明書』(*1)

本製品の仕様や基本的な取り扱い方法、トラブルの対処方法について説明しています。必ずお読みください。

『BIOS 説明書』

添付のドライバ CD 内に PDF データとして格納されています(*2)。

OS のインストール方法や、ハードウェア環境を設定するためのプログラムである BIOS Setup の仕様と操作方法について説明しています。OS をインストールする際や、ご購入時にあらかじめ設定されている BIOS の設定を変更する場合にお読みください。

『RAS 制御 API 説明書（Windows 編（本書）、Linux 編）』

添付のドライバ CD 内に PDF データとして格納されています(*2)。

各種 API の仕様について説明しています。

『RAS 制御チュートリアル（Windows 編、Linux 編）』

添付のドライバ CD 内に PDF データとして格納されています(*2)。

具体的なプログラムを例に、本製品を使ったシステム構築について説明しています。

*1）取扱説明書のみ、英語版が存在します。

*2）PDF 形式のファイルをご覧いただく際は、Adobe® Reader® が必要です。
Adobe® Reader® は、アドビシステムズ社のサイトから無償でダウンロードできます。

●本書の構成について

第 1 章　概要

本製品の概要について説明しています。

第 2 章　AR_DIO.DLL リファレンス

AR_DIO.DLL について説明しています。

第 3 章　システム信号制御 リファレンス

システム信号制御について説明しています。

付録

仕様や留意事項について説明します。

●略称

本書では、以下の用語について省略して表記する場合がありますので、ご了承ください。

製品名称	略称
Microsoft® Windows® XP Professional operating system	Windows® XP
Microsoft® Windows® 2000 Professional operating system	Windows® 2000
Red Hat® Enterprise Linux v3 Red Hat® Enterprise Linux v4	Linux

このページは空白です。

目次

はじめに i
本書の読み方 ii
●マニュアル体系 ii
●本書の構成について iii
●略称 iii

目次 v

第 1 章　概要 1
1.1　特長 1
1.2　機能 1
1.3　動作環境 2
1.4　必要なファイルについて 2

第 2 章　AR_DIO.DLL リファレンス 3
2.1　基本操作 3
（1） DioOpen（ ）関数 3
（2） DioClose（ ）関数 3
2.2　入出力操作 4
（1） DioInput（ ）関数、DioOutput（ ）関数 4
（2） DioInputAll（ ）関数、DioOutputAll（ ）関数 4
2.3　バックグラウンド（BG）処理 6
（1） DioSetBack（ ）関数 6
（2） DioSetBackPulse（ ）関数 7
（3） DioSetBackCustom（ ）関数 8
（4） DioSetBackPatternTrg（ ）関数 10
（5） DioSetBackTimerTrg（ ）関数 11
（6） DioStartBack（ ）関数 12
（7） DioStartBackAll（ ）関数 13
（8） DioStopBack（ ）関数 14
（9） DioStopBackAll（ ）関数 15
（10） DioGetBackStatus（ ）関数 16
（11） DioRemoveBack（ ）関数 18
（12） DioCleanupBack（ ）関数 18

2.4 デジタルフィルタ制御 19
（1） DioGetFilter（ ）関数、DioSetFilter（ ）関数 19

第 3 章 システム信号制御リファレンス 21
3.1 基本操作 21
（1） SystemOpen（ ）関数 21
（2） SystemClose（ ）関数 21
3.2 アラーム制御 22
（1） SetAlarm（ ）関数 22
（2） GetAlarm（ ）関数 23
（3） ClearAlarm（ ）関数 23
（4） AlarmIntCustomize（ ）関数 24
3.3 WatchdogTimeout 制御 25
（1） StartWDT（ ）関数 25
（2） ContinueWDT（ ）関数 26
（3） StopWDT（ ）関数 27
（4） CustomizeWDT（ ）関数 27
3.4 リセット制御 29
（1） GetResetStatus（ ）関数 29
（2） ClearResetStatus（ ）関数 30
3.5 外部リセット機能 31
（1） ExternalReset（ ）関数 31

付録 エラーコード一覧 33

第 1 章 概要

この章では、本製品の概要について説明します。

1.1 特長

AR シリーズでは、アプリケーションから次のような多彩な機能を利用できるためのライブラリとして、各種 API を提供しています。

- DIO 制御機能
- システム信号制御機能（温度、電圧、ファン監視機能は除く）

本書では、DIO 制御を行うためのライブラリと、システム制御を行うライブラリの使用について説明します。API 部は DIO の機能を用いる場合と、システム全体を管理する場合の 2 種類に分けて構成されています。この 2 種類の API 関数群を用いることにより、AR シリーズの DIO 機能とシステム管理機能を利用することができます。

1.2 機能

本製品には、以下の機能があります。

DIO 制御部

- DIO 機能 ：DIO 接点に対して入出力を行います。
- バックグラウンド機能 ：定期的な出力処理をバックグラウンドで行います。
- デジタルフィルタ機能 ：デジタルフィルタ制御を行います。

システム信号制御部

- アラーム機能 ：アラームの各種制御を行います。
- WatchDogTimeout 機能 ：ウォッチドッグタイマの各種制御を行います。
- リセット情報取得機能 ：各種リセットに対する制御を行います。
- リセット出力 ：リセット出力 (RSTOUT) をアプリケーションから発信します。

1.3　動作環境

本製品を使用するためには、以下の動作環境が必要です。

OS：Windows® XP Professional SP2
Windows® 2000 SP4
※日本語環境のみ
必要なファイル：Windows/System32 インストールディレクトリ
お使いのアプリケーションソースフォルダ

1.4　必要なファイルについて

本製品を使用するためには、以下のファイルが必要です。

- Windows/System32 インストールディレクトリ
  - AR_DIO.DLL
    DIO 制御用ライブラリファイル
  - AR_SYSTEM.DLL
    システム信号制御用ライブラリファイル
- お使いのアプリケーションソースフォルダ
  - AR_DIO.h
  - AR_SYSTEM.h
  - AR_DIO.lib
  - AR_SYSTEM.lib

第 2 章　AR_DIO.DLL リファレンス

この章では、AR_DIO.DLL リファレンスについて説明します。

2.1　基本操作

ボードの DIO 制御部にアクセスするための関数について説明します。

（1） DioOpen（ ）関数

【機能説明】

DIO デバイスをオープンします。

【呼び出し形式】

HANDLE　DioOpen(char *Name, DWORD DioMaxNum, BOOL　LogEnable)

【パラメタ】

Name	：	任意の名前を指定します。
DioMaxNum	：	DIO の最大接点数を指定します。 1 ～ 16
LogEnable	：	常に FALSE としてください。

【復帰値】

ハンドル値	：	正常終了アクセス要求を行うためのデバイスハンドルを返します。
NULL	：	異常終了

（2） DioClose（ ）関数

【機能説明】

DIO デバイスをクローズします。

【呼び出し形式】

INT DioClose(HANDLE hDevH)

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了

2.2 入出力操作

DIO ボードに対して、基本的なオン／オフ操作を行います。

（1） DioInput（ ）関数、DioOutput（ ）関数

【機能説明】

指定接点の状態を取得（入力）、または設定（出力）します。

【呼び出し形式】

INT DioInput(HANDLE hDevH, DWORD No, PDWORD Rtn_Value)
INT DioOutput(HANDLE hDevH, DWORD No, DWORD Set_Value)

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	入力（出力）接点の番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
Rtn_Value	：	入力データを格納するバッファです。 0：入力接点がオフ状態である事を示しています。 1：入力接点がオン状態である事を示しています。
Set_Value	：	出力データを格納するバッファです。 0：出力接点をオフ状態にします。 1：出力接点をオン状態にします。

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

（2） DioInputAll（ ）関数、DioOutputAll（ ）関数

【機能説明】

全入力接点の状態を取得、または全出力接点を設定します。

【呼び出し形式】

INT DioInputAll(HANDLE hDevH, PDWORD Rtn_Value)
INT DioOutputAll(HANDLE hDevH, PDWORD Set_OnValue, PDWORD Set_OffValue)

【パラメタ】

hDevH	:	DioOpen によって返されたデバイスハンドルを指定します。
Rtn_Value	:	全入力データを格納するバッファです。
Set_OnValue	:	出力接点をオンにする全接点のデータを設定するバッファの配列です。 0：何もしない 1：オン値をセットする
Set_OffValue	:	出力接点をオフにする全接点のデータを設定するバッファの配列です。 0：何もしない 1：オフ値をセットする

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了

ADS_UNSUITABLE_PROCCESS (-10)

詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

Set_OnValue	Set_OffValue	
0	0	何もしない
0	1	オフ値をセットする
1	0	オン値をセットする
1	1	ソフトエラーとする

【サンプルプログラム】

```
DWORD OnValue[16] , OffValue[16];
for (int i=0; i<16; i++) {
    OnValue[i] = 0;
    OffValue[i] = 0;
}
OnValue[1] = 1;   // DIO2 番を ON
OnValue[4] = 1;   // DIO5 番を ON
OffValue[11] = 1; // DIO12 番を OFF
OffValue[15] = 1; // DIO16 番を OFF
DioOutputAll(hDevH, OnValue, OffValue); // 対象の 4 接点を同時に出力実行
上記配列指定により 2 番、5 番がオン。12 番、16 番がオフ。
他の点は現状維持（オン→オン または オフ→オフ）となります。
```

2.3 バックグラウンド（BG）処理

バックグラウンドにて出力接点のオン／オフ操作を定期的に行わせます。
登録・開始・停止・削除を制御する事で DO 出力の自動実行化が可能となります。
アプリケーション終了後も自動実行は引き続き行われます。

（1） DioSetBack（ ）関数

【機能説明】

簡易的なバックグラウンド処理を即座に実行します。
指定接点に対し、一定周期でオン／オフを行うバックグラウンド処理を行います。
この処理は停止を行わない限り無限に処理を続きます。
本関数で指定した端子において，DioSetBackPulse または DioSetBackCustom によりバックグラウンド処理が登録されていた場合，設定が上書きされます。

【呼び出し形式】

INT DioSetBack(HANDLE hDevH, DWORD No, DWORD Cycle)

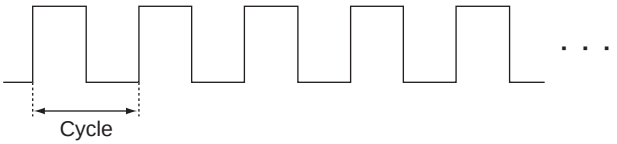
【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	出力接点の開始番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
Cycle	：	オン／オフの周期を指定します 20 ～ 1000000［msec］

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】



オン／オフを一度終えた時点で 1 サイクルとなります。
本関数は登録と同時にバックグラウンド処理は即座に開始されます。

（2） DioSetBackPulse（ ）関数

【機能説明】

簡易的なバックグラウンド処理を即座に実行します。指定接点に対し、指定時間幅のパルス出力を行う処理を行います。
本関数で指定した端子において，DioSetBack または DioSetBackCustom によりバックグラウンド処理が登録されていた場合，設定が上書きされます。

【呼び出し形式】

```
HANDLE　DioSetBackPulse(HANDLE hDevH, DWORD No,
                        DWORD PulseWidth, DWORD Polarity)
```

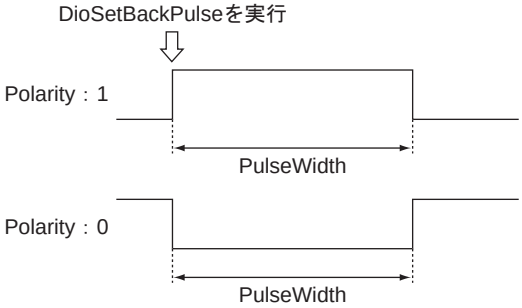
【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	出力接点の開始番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
PulseWidth	：	パルス幅を指定します 10 ～ 1000000［msec］
Polarity	：	極性フラグを指定します。 0：オン→オフのパルスを出力 1：オフ→オンのパルスを出力

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】



PulseWidth の期間だけパルスを出力します。出力は 1 回のみです。

（3） DioSetBackCustom（ ）関数

【機能説明】

指定接点に対し、任意の周期で、任意の回数だけ繰り返し処理を行うバックグラウンド処理を登録します。本関数で指定した端子において，DioSetBack または DioSetBackPulse によりバックグラウンド処理が登録されていた場合，設定が上書きされます。

【呼び出し形式】

```
INT DioSetBackCustom(HANDLE hDevH, DWORD No, PDWORD pBuffer,
                     DWORD ValueNum, DWORD LoopCount)
```

【パラメタ】

hDevH	:	DioOpen によって返されたデバイスハンドルを指定します。
No	:	出力接点の開始番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
pBuffer	:	1 周期のパターンを格納したバッファのアドレスを指定します。
ValueNum	:	1 周期のパターンを格納したバッファのデータ数を指定します。 1 ～ 1024
LoopCount	:	指定したパターンを繰り返す回数を指定します。 1 ～ 1000000：実行回数 0：無限に行います

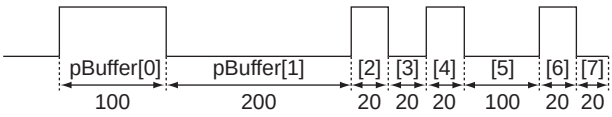
【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

本関数による登録時に、指定番号のバックグラウンド処理が稼動中であった場合、ソフトエラーとして登録は行いません。一度停止関数を行ってから再度本関数にて登録処理を行ってください。

単位 [msec」
pBuffer[0] = 100
pBuffer[1] = 200
pBuffer[2] = 20
pBuffer[3] = 20
pBuffer[4] = 20
pBuffer[5] = 100
pBuffer[6] = 20
pBuffer[7] = 20



オン状態のパルス幅、オフ状態のパルス幅をそれぞれ 1 バッファごとに格納して、指定したバッファ数を通じて 1 サイクルとします。

【サンプルプログラム】

```
DWORD pBuffer[8];
pBuffer[0] = 100
pBuffer[1] = 200
pBuffer[2] = 20
pBuffer[3] = 20
pBuffer[4] = 20
pBuffer[5] = 100
pBuffer[6] = 20
pBuffer[7] = 20

DioSetBackCustom(hDevH, 3, pBuffer, 8, 100); // 3 番に上記パターンを 100 回行う。
```

（4） DioSetBackPatternTrg（ ）関数

【機能説明】

指定接点に対し、バックグラウンドを開始するための入力接点の条件を設定します。

【呼び出し形式】

```
INT DioSetBackPatternTrg(HANDLE hDevH, DWORD No, PDWORD ON_Pattern,
                         PDWORD OFF_Pattern)
```

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	自動開始条件を設定する出力接点番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
ON_Pattern	：	入力接点 ON 状態で検知するトリガの配列を格納します。 0：トリガ設定無し 1：オン・トリガを有効
OFF_Pattern	：	入力接点 OFF 状態で検知するトリガの配列を格納します。 0：トリガ設定無し 1：オフ・トリガを有効

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【サンプルプログラム】

```
DWORD OFF_Pattern[16];
DWORD ON_Pattern[16];
for(int i=0; i<16; i++) {
    OFF_Pattern[i] = 0;
    ON_Pattern[i] = 0;
}
ON_Pattern[1] = 1;   // 2 番目が ON 状態
ON_Pattern[4] = 1;   // 5 番目が ON 状態
ON_Pattern[7] = 1;   // 8 番目が ON 状態
OFF_Pattern[15] = 1; // 16 番目が OFF 状態
DioSetBackPatternTrg(hDevH, 1, ON_Pattern, OFF_Pattern);//1 番の自動条件開始登録
```

上記例のような配列指定を行う事で DIO のオフ／オン状態をトリガ設定とする事ができます。
上記例では DIO2 番、5 番、8 番がオン状態でなおかつ 16 番がオフ状態の時に 1 番のバックグラウンド処理が開始されます。

【補足説明】

本関数の指定を行うことにより、バックグラウンド処理開始後にドライバにおいて 10ms 間隔で入力接点を監視し、トリガ条件が成立した時にバックグラウンド動作を開始します。
トリガ条件の入力信号につきましては、10ms より十分長い信号が入力されるようにしてください。10ms よりも短い信号が入力された場合、トリガ条件が成立していたとしても、バックグラウンド処理が開始されないことがあります。

（5） DioSetBackTimerTrg（ ）関数

【機能説明】

指定接点に対し、バックグラウンドを開始するディレイタイムを設定します。

【呼び出し形式】

INT DioSetBackTimerTrg(HANDLE hDevH, DWORD No, DWORD Delay)

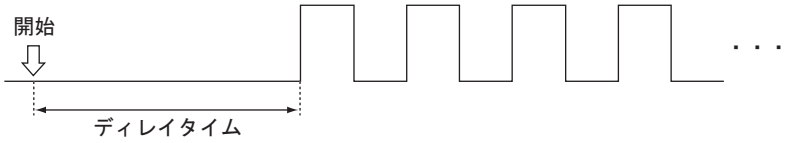
【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	ディレイ設定を行う出力接点の番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
Delay	：	ディレイ時間を数字で指定します。 10 ～ 1000000[msec]

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】



（6） DioStartBack（ ）関数

【機能説明】

指定接点に登録されているバックグラウンド処理を初めから開始します。

【呼び出し形式】

INT DioStartBack(HANDLE hDevH, DWORD No)

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	開始させたい接点番号

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 ADS_UNSUITABLE_PROCESS (-10) 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

指定した接点に登録されたパターンで最初からバックグラウンド処理を開始します。
指定した接点においてバックグラウンド処理が実行中の場合は異常終了し、現在動作している処理を継続します。
バックグラウンド処理が実行中の時に、最初からバックグラウンド処理を実行させる場合は、DioStopBack 関数または DioStopBackAll 関数で一旦バックグラウンド処理を停止してから、本関数を実行してください。
DioSetBackPatternTrg 関数により入力接点のオフ／オン状態をトリガ条件に設定している場合には、トリガ条件が成立するまでバックグラウンド処理は実行されません。
複数接点のバックグラウンド処理を同時に開始させる場合は、
DioStartBackAll 関数を使用してください。

（7） DioStartBackAll（ ）関数

【機能説明】

指定した複数接点のバックグラウンド処理を同時に開始します。

【呼び出し形式】

INT DioStartBackAll(HANDLE hDevH, PDWORD StartFlag)

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
StartFlag	：	バックグラウンド処理開始を指定する全接点データのアドレスを指定します。 0：何もしない 1：バックグラウンド処理を開始

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了

ADS_UNSUITABLE_PROCESS (-10)

詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

各接点に登録されたパターンで最初からバックグラウンド処理を同時に開始します。
指定した接点においてバックグラウンド処理が実行中の場合は異常終了し、現在動作している処理を継続します。
バックグラウンド処理が実行中の時に、最初からバックグラウンド処理を実行させる場合は、DioStopBack 関数または DioStopBackAll 関数で一旦バックグラウンド処理を停止してから、本関数を実行してください。
DioSetBackPatternTrg 関数により入力接点のオフ／オン状態をトリガ条件に設定している場合には、トリガ条件が成立するまでバックグラウンド処理は実行されません。

【サンプルプログラム】

```
DWORD StartFlag[16];
INT result;

for(int i=0; i<16; i++) {
StartFlag[i] = 0;
}
StartFlag[1]  = 1; // DIO_2 番目
StartFlag[4]  = 1; // DIO_5 番目
StartFlag[15] = 1; // DIO_16 番目

result = DioStartBackAll(hDevH, StartFlag); // 同時にスタート実行

if (result == -10) {
DioStopBackAll(hDevH, StartFlag); // 同時停止関数により 2,5,16 番を停止

    result = DioStartBackAll(hDevH, StartFlag);
    if (result != 0)
    {
        MessageBox("BackGround Error.", 0, MB_OK);
    }
}
```

上記のような配列指定を行うことで 2,5,16 番目のバックグラウンド処理が同時にスタートします。また本サンプルでは、2,5,16 番のいずれかが稼動中である場合、2,5,16 番を一度停止させ、再度開始を試みています。

（8） DioStopBack（ ）関数

【機能説明】

指定接点のバックグラウンド処理を停止します。
本関数ではバックグラウンド登録情報は削除されません。

【呼び出し形式】

INT DioStopBack(HANDLE hDevH, DWORD No)

【パラメタ】

hDevH	:	DioOpen によって返されたデバイスハンドルを指定します。
No	:	停止させたいバックグラウンド処理の接点番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

本関数から 1 点ごとに停止させる処理を表記しても、全接点同時停止を行う際には遅れが生じる可能性があります。緊急で全接点停止を行う場合は DioStopBackAll 関数を用いてください。

（9） DioStopBackAll（ ）関数

【機能説明】

指定した複数接点のバックグラウンド処理を同時に停止させます。
本関数ではバックグラウンド登録情報は削除されません。

【呼び出し形式】

INT DioStopBackAll(HANDLE hDevH, PDWORD StopFlag)

【パラメタ】

hDevH	:	DioOpen によって返されたデバイスハンドルを指定します。
StopFlag	:	バックグラウンド処理停止を指定する全接点データのアドレスを指定します。 0：何もしない 1：バックグラウンド処理を停止

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

複数指定した接点に既に停止済みの番号が含まれていても本関数は成功します。

【サンプルプログラム】

```
DWORD StopFlag[16];

for(int i=0; i<16; i++) {
StartFlag[i] = 0;
}
StopFlag[1]  = 1; // DIO_2 番目
StopFlag[4]  = 1; // DIO_5 番目
StopFlag[15] = 1; // DIO_16 番目

DioStartBackAll(hDevH, StopFlag); // 同時停止を実行
```

上記配列により 2,5,16 番を同時に停止させます。

（10） DioGetBackStatus（ ）関数

【機能説明】

指定接点のバックグラウンド処理情報を取得します。
稼動中のフラグ、ループ回数、トリガ設定の有無が確認できます。

【呼び出し形式】

```
INT DioGetBackStatus(HANDLE hDevH, DWORD No, PDWORD ValueNum,
                     PDWORD MaxLoop, PDWORD Count, PDWORD
                     TriggerFlag, PDWORD Execute)
```

【パラメタ】

hDevH	:	DioOpen によって返されたデバイスハンドルを指定します。
No	:	情報を取得するバックグラウンド処理の接点番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）
ValueNum	:	登録されている 1 周期のパターンを格納したバッファのデータ数が格納されます。
MaxLoop	:	登録されているバックグラウンド処理のパターンを繰り返す回数が格納されます。
Count	:	現在実行中のバックグラウンド処理のループ回数が格納されます。
TriggerFlag	:	DioSetBackPatternTrg 関数で登録されたトリガ設定の状態が格納されます。 0：トリガ設定無 1：トリガ設定有
Execute	:	バックグラウンド処理の稼動状態が格納されます。 0：停止状態 1：稼動中

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

MaxLoop 値が 0 の場合、無限ループが指定されていることを示しています。
Count 値は、DioStartBack 関数、DioStartBackAll 関数、DioRemoveBack 関数、および DioCleanupBack 関数により 0 に初期化され、1 周期のパターンが終了するごとにインクリメントされます。

（11） DioRemoveBack（ ）関数

【機能説明】

指定接点の登録されているバックグラウンド処理の登録情報を削除します。
バッファ設定値、最大ループ数、ループ回数、トリガ設定、稼動フラグを初期化します。
本関数は DioStopBack 関数の機能を含みます。

【呼び出し形式】

INT DioRemoveBack(HANDLE hDevH, DWORD No)

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。
No	：	削除させたいバックグラウンド処理の接点番号を数字で指定します。 1 ～最大接点番号（DioMaxNum）

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

（12） DioCleanupBack（ ）関数

【機能説明】

全接点の登録されているバックグラウンド処理を同時に停止し、全接点の登録情報を同時に削除します。

【呼び出し形式】

INT DioCleanupBack(HANDLE hDevH)

【パラメタ】

hDevH	：	DioOpen によって返されたデバイスハンドルを指定します。

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

2.4 デジタルフィルタ制御

DIO のチャタリング現象が著しい場合、デジタルフィルタを設ける事でチャタリング現象を防ぐことができます。

（1） DioGetFilter（ ）関数、DioSetFilter（ ）関数

【機能説明】

ポート（8 接点）ごとのデジタルフィルタを取得、設定します。

【呼び出し形式】

INT DioGetFilter(HANDLE hDevH, char *Key, PDWORD Rtn_Filter)
INT DioSetFilter(HANDLE hDevH, char *Key, DWORD Set_Filter)

【パラメタ】

hDevH	:	DioOpen によって返されたデバイスハンドルを指定します。
Key	:	どのポートを選択するかを以下の文字列によって指定します。 DIO_IN1_8　DIO_IN9_16
Rtn_Filter	:	現在のフィルタ設定値を格納するバッファを指定します。
Set_Filter	:	フィルタの設定値を 18 段階の数字で指定します。 0：デジタルフィルタ無効 1 ～ 18：1us ～ 131.072ms

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

デジタルフィルタの制御値は 2 の Set_Filter 剰で増加することを仕様としています。

1	---	1us	7	---	64us	13	---	4.096msec
2	---	2us	8	---	128us	14	---	8.192msec
3	---	4us	9	---	256us	15	---	16.384msec
4	---	8us	10	---	512us	16	---	32.768msec
5	---	16us	11	---	1.024msec	17	---	65.536msec
6	---	32us	12	---	2.048msec	18	---	131.072msec

このページは空白です。

第３章　システム信号制御リファレンス

この章では、システム信号制御リファレンスについて説明します。

3.1　基本操作

ボードのシステム部にアクセスするための関数について説明します。

（1） SystemOpen（ ）関数

【機能説明】
　　システム管理機能をオープンします。

【呼び出し形式】
　　HANDLE　　SystemOpen()

【パラメタ】
　　なし

【復帰値】

ハンドル値	：	正常終了 アクセス要求を行うためのデバイスハンドルを返します。
NULL	：	異常終了

（2） SystemClose（ ）関数

【機能説明】
　　システム管理機能をクローズします。

【呼び出し形式】
　　INT SystemClose(HANDLE hDevH)

【パラメタ】

hDevH	：	SystemOpen によって返されたデバイスハンドルを指定します。

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了

3.2 アラーム制御

アラーム部の設定を行います。2 種類のアラーム信号の検知方法、ユーザ指定の出力タイプなどを指定することでシステムを構築することができます。

【補足説明】

本機能は、EmbedWare/SysMon DIO オプションが使用します。一般のアプリケーションから使用することはできません。

（1） SetAlarm（ ）関数

【機能説明】

指定されたアラーム信号の出力タイプを設定します。

【呼び出し形式】

INT SetAlarm(HANDLE hDevH, char *Key1, char *Set_Type)

【パラメタ】

hDevH	：	SystemOpen によって返されたデバイスハンドルを指定します。
Key1	：	アラーム識別番号を以下の文字列で指定します。 ALARM1 , ALARM4
Set_Type	：	アサート時の出力タイプを以下の文字列によって指定します。
ALM_SYS_RESET	：	システムリセットを実行します
ALM_GP_RESET	：	リセット出力信号を発信します
ALM_OUT	：	アラームイベントを設定します
ALM_NONE	：	何も行いません

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

ALM_NONE を指定した場合、システムリセット実行、リセット出力信号発信、およびアラームイベント設定は行われません。

（2） GetAlarm（ ）関数

【機能説明】

アラームイベントの情報を取得します。

【呼び出し形式】

INT GetAlarm(HANDLE hDevH, char *Key, PDWORD Rtn_Value)

【パラメタ】

hDevH	:	SystemOpen によって返されたデバイスハンドルを指定します。
Key	:	アラーム識別番号を以下の文字列で指定します。 ALARM1 , ALARM4
Rtn_Value	:	アラームイベントの情報を格納するバッファです。 0：イベント無 1：イベント発生済

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

（3） ClearAlarm（ ）関数

【機能説明】

指定したアラームイベント情報をクリアします。

【呼び出し形式】

INT ClearAlarm(HANDLE hDevH, char *Key)

【パラメタ】

hDevH	:	SystemOpen によって返されたデバイスハンドルを指定します。
Key	:	アラーム識別番号を以下の文字列で指定します。 ALARM1 , ALARM4

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

（4） AlarmIntCustomize（ ）関数

【機能説明】

各アラームの割り込み検知方法を設定します。

【呼び出し形式】

INT AlarmIntCustomize(HANDLE hDevH, char *Key, DWORD Set_IntType)

【パラメタ】

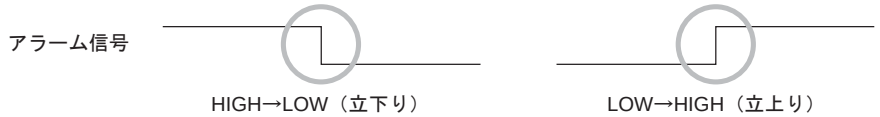
hDevH	：	SystemOpen によって返されたデバイスハンドルを指定します。
Key	：	アラーム識別を以下の文字列で指定します。 ALARM1 , ALARM4
Set_IntType	：	割り込み検知方法を指定します。 0：HIGH -> LOW （立下り）1：LOW （レベル） 2：LOW -> HIGH （立上り）3：HIGH（レベル）

【復帰値】

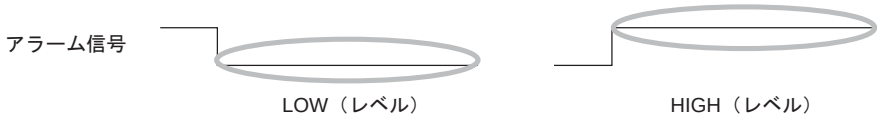
0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

割り込み検知方法のデフォルト値は 1（LOW レベル検知）です。
割り込み検知『HIGH->LOW』『LOW->HIGH』の場合は以下の時 Alarm 検知とみなします。
アラーム検知は一度しか行われません。



割り込み検知『LOW』『HIGH』の場合は以下の状態で常にアラーム検知とみなします。
アラーム信号の出力タイプがアラームイベント情報設定の場合、ClearAlarm 関数によりアラームイベントをクリアしても、アラーム信号が検知レベルになっていれば、再度アラームイベントが設定されます。



3.3 WatchdogTimeout 制御

ウォッチドックタイマ機能を有効とすることによって、OS またはアプリケーションプログラムがハングアップした場合、強制的な再起動を行う事ができます。

【補足説明】
本機能を使用する場合、EmbedWare/SysMon DIO オプションのウオッチドッグタイマ機能を使用しないでください。

（1） StartWDT（ ）関数

【機能説明】

ウォッチドッグタイマの機能を有効にします。
タイムアウト時はシステムリセット、リセット出力、両方のリセットを発生させる事ができます。

【呼び出し形式】

INT StartWDT(HANDLE hDevH, char *Set_Type, DWORD Timer)

【パラメタ】

hDevH	：	SystemOpen によって返されたデバイスハンドルを指定します。
Set_Type	：	タイムアウト時の出力タイプを以下の文字列によって指定します。 WDT_SYS_RESET：システムリセットを実行します WDT_GP_RESET：リセット出力信号を発信します WDT_SYS_AND_GP：リセット出力信号を発信し、システムリセットを実行します WDT_OUT：ウォッチドックタイムアウト出力信号を発信します WDT_NONE ：何も行いません
Timer	：	タイムアウト時間を設定します。 10 ～ 250000 [msec]

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

タイムアウト時間は、1[msec] の位の値は切り捨てられます。
また、最上位桁からみて 2 桁下以降の指定に対しては切り捨てとする事を仕様とします。

42	⇒	40[msec]	
554	⇒	550[msec]	
7631	⇒	7600[msec]	
35117	⇒	35000[msec]	（35[sec]）
234789	⇒	234000[msec]	（234[sec]）

（2） ContinueWDT（ ）関数

【機能説明】

ウォッチドッグタイマを更新し、タイムアウトさせず継続します。
この関数は StartWDT を行ったあとから有効となります。

【呼び出し形式】

INT ContinueWDT(HANDLE hDevH)

【パラメタ】

hDevH : SystemOpen によって返されたデバイスハンドルを指定します。

【復帰値】

0 : 正常終了

ADS_SUCCESS(0) : 正常終了

負の値 : 異常終了

ADS_UNSUITABLE_PROCESS (-10)

詳細はエラーコード一覧（→ 33 ページ）を参照してください。

（3） StopWDT（ ）関数

【機能説明】

動作中のウォッチドッグタイマを停止させ、監視を終了します。
この関数は StartWDT を行ったあとから有効となります。

【呼び出し形式】

INT StopWDT(HANDLE hDevH)

【パラメタ】

hDevH : SystemOpen によって返されたデバイスハンドルを指定します。

【復帰値】

0 : 正常終了

ADS_SUCCESS(0) : 正常終了

負の値 : 異常終了

ADS_UNSUITABLE_PROCESS (-10)

詳細はエラーコード一覧（→ 33 ページ）を参照してください。

（4） CustomizeWDT（ ）関数

【機能説明】

StartWDT により指定したウォッチドッグタイムアウト（以下、「1 回目のタイムアウト」とします）が発生した後で、再度ウォッチドッグタイムアウトが発生した場合（以下、「2 回目のタイムアウト」とします）に行う動作を本関数で指定します。
この関数は StartWDT を行う前に実行してください。

【呼び出し形式】

INT CustomizeWDT (HANDLE hDevH, char *Set_Type)

【パラメタ】

hDevH	：	SystemOpen によって返されたデバイスハンドルを指定します。
Set_Type	：	2 回目タイムアウト時の出力タイプを以下の文字列によって指定します。 WDT_SYS_RESET：システムリセットを実行します WDT_GP_RESET：リセット出力信号を発信します WDT_SYS_AND_GP：リセット出力信号を発信し、システムリセットを実行します WDT_OUT：ウォッチドックタイムアウト出力信号を発信します。 WDT_NONE ：何も行いません

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

1 回目のタイムアウトが発生してから StartWDT 関数で指定されたタイムアウト時間が経過した時に、2 回目のタイムアウトが発生します。

StartWDT 関数の出力タイプでシステムリセット発行を指定した場合や他の要因でシステムリセットが発生した場合は、ウォッチドックタイマが初期化されるため、2 回目のタイムアウトは発生しません。

3.4 リセット制御

各種リセット信号を情報取得・制御する事ができます。
リセット信号を把握する事でシステム全体の異常発覚時に前後検証を行う事ができます。

（1） GetResetStatus（ ）関数

【機能説明】

リセット要因を取得します。

【呼び出し形式】

INT GetResetStatus(HANDLE hDevH, PDWORD Rtn_Value)

【パラメタ】

hDevH : SystemOpen によって返されたデバイスハンドルを指定します。

Rtn_Value : 各々のリセットの状態を取得します。バッファ内には以下に示すようなビット情報が格納されます。

外部リセット部の要因

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
POWER	PCI	GP	WDT	ALARM4	0	0	ALARM1

システムリセット部の要因

Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
POWER	SYSTEM	0	WDT	ALARM4	0	0	ALARM1

Bit131	・・・	・・・	・・・	・・・	・・・	・・・	Bit16
0	0	0	0	0	0	0	0

POWER : 電源オフ／オン状態を検知

PCI : PCI バスリセット発生を検知

GP : DIO 部リモートリセット入力（DIO_RSTIN）を検知

WDT : ウォッチドッグによるリセット発生を検知

ALARM1 , ALARM4 : アラームによるリセット発生を検知

SYSTEM : システムリセット発生を検知

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

ALARM1、ALARM4、WDT、GP、PCI、および SYSTEM のリセット要因は、電源オフ・オンが行われたあとは 0 に初期化されます。
ハイバネーションおよびサスペンド／レジュームされた場合においても、リセット要因が 0 に初期化されます。

（2） ClearResetStatus（ ）関数

【機能説明】

指定部分のリセットステータス値を 0 に初期化します。

【呼び出し形式】

INT ClearResetStatus(HANDLE hDevH, char *Key)

【パラメタ】

hDevH	：	SystemOpen によって返されたデバイスハンドルを指定します。
Key	：	初期化するリセット要因を選択します。 ALL：全要因 SYS_ALL：システムリセット部の全要因 GP_ALL：外部リセット部の全要因

【復帰値】

0	：	正常終了
ADS_SUCCESS(0)	：	正常終了
負の値	：	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

【補足説明】

個別指定クリアを行いたい場合は以下の項目を選択してください。

『システムリセット部』	『外部リセット部』		
SYS_POWER	GP_POWER	:	電源オフ／オン要因
SYS_SYSTEM	GP_PCI	:	システムリセット要因
	GP_GP	:	GP リセット要因
SYS_WDT	GP_WDT	:	WDT 要因
SYS_ALARM1	GP_ALARM1	:	アラーム 1 要因
SYS_ALARM4	GP_ALARM4	:	アラーム 4 要因

3.5 外部リセット機能

リセット出力信号をアプリケーションから制御します。

（1） ExternalReset（ ）関数

【機能説明】

リセット出力信号（RSTOUT）を発信します。
システムリセットは行われません。

【呼び出し形式】

INT ExternalReset(HANDLE hDevH)

【パラメタ】

hDevH	:	SystemOpen によって返されたデバイスハンドルを指定します。

【復帰値】

0	:	正常終了
ADS_SUCCESS(0)	:	正常終了
負の値	:	異常終了 詳細はエラーコード一覧（→ 33 ページ）を参照してください。

このページは空白です。

付録 エラーコード一覧

定数	値	説明
ADS_SUCCESS	0	正常終了
ADS_ERROR	(-1)	システムエラーが発生
ADS_PRMERR	(-2)	指定パラメタに誤りがある。
ADS_INVALID_HANDLE	(-3)	デバイスのオープンに失敗
ADS_NODATA	(-9)	引数 Key で指定された項目がない。
ADS_UNSUITABLE_PROCESS	(-10)	関数の実行手順に誤りがある。

このページは空白です。

エンベデッドコンピュータ AR2000 シリーズ
フロントオペレーションモデル
RAS 制御 API 説明書（Windows 編）

P3XU-E501-02Z0

発 行 日　2007 年 5 月

発行責任　株式会社 PFU

●本書の内容は、改善のため事前連絡なしに変更することがあります。
●本書に記載されたデータの使用に起因する、第三者の特許権およびその他の権利の侵害については、当社はその責を負いません。
●無断転載を禁じます。